CPS104 Computer Organization

Lecture 7

MIPS Instruction Set Architecture; MIPS Assembler

September 14 , 2009

Gershon Kedem

©GK Fall 2009

Administratrivia

- Programming Puzzle-2 is on the Web, see:
  `kedem.cs.duke.edu/cps104/ppz/PPZ2.html`
- Due: September 16 (11:59 pm).
- Submit the `*.java` file to `cps104/PPZ2`
- Use the Eclipse system to submit the file.
- Help is on-line.
- Reading: Chapter 2.
- Reading: Appendix B
  - Also available on:. . . `cps104/Handouts/spim_appendix.pdf`

©GK Fall 2009

Review: Basic ISA Classes

Accumulator:

1 address	add A	acc ← acc + mem[A]
1+x address	addx A	acc ← acc + mem[A + x]

Stack:

0 address	add	tos ← tos + next (JAVA VM)

General Purpose Register:

2 address	add A B	A ← A + B
3 address	add A B C	A ← B + C

Load/Store:

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

©GK Fall 2009

Review: LOAD / STORE ISA

- **Instruction set:**
  - **add, sub, mult, div, … only on operands in registers**
  - **ld, st, to move data from and to memory,**
    The only way to access memory

Example: a*b - (a+c*b)

	r1,	r2,	r3
ld r1, c	2,	?,	?
ld r2, b	2,	3,	?
mult r1, r1, r2	6,	3,	?
ld r3, a	6,	3,	4
add r1, r1, r3	10,	3,	4
mult r2, r2, r3	10,	12,	4
sub r3, r2, r1	10,	12,	2

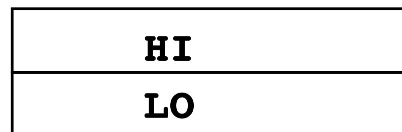
a	4
b	3
c	2

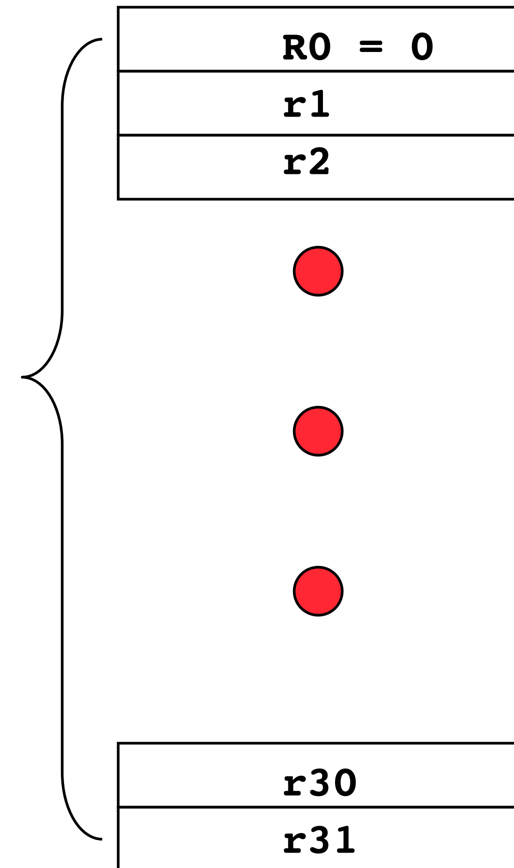
7 instructions

©GK Fall 2009

Review: MIPS Registers

- Registers: fast memory, Integral part of the CPU.
- Programmable storage 2^{32} bytes
- 31 x 32-bit GPRs (R0 = 0)
- 32 x 32-bit FP regs (paired for DP)
- 32-bit HI, LO, PC

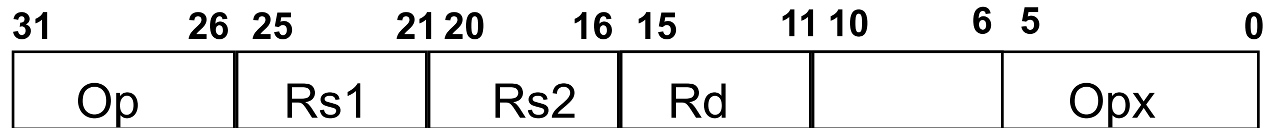




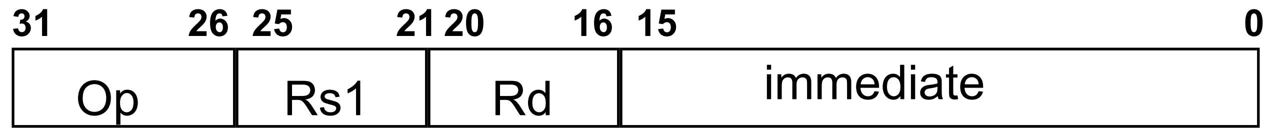
©GK Fall 2009

Example: MIPS Instructions' Formats

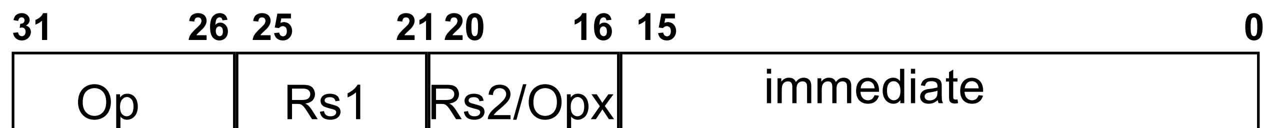
Register-Register



Register-Immediate



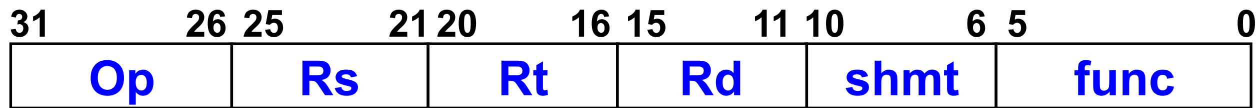
Branch



Jump / Call

©GK Fall 2009

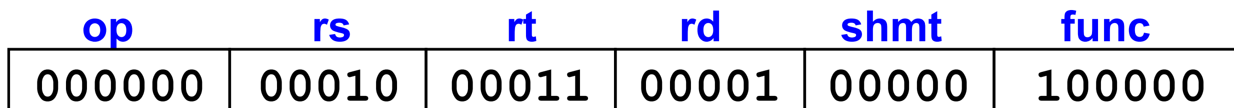
R Type: <OP> rd, rs, rt



op	a 6-bit operation code.
rs	a 5-bit source register.
rt	a 5-bit target (source) register.
rd	a 5-bit destination register.
shmt	a 5-bit shift amount.
func	a 6-bit function field.

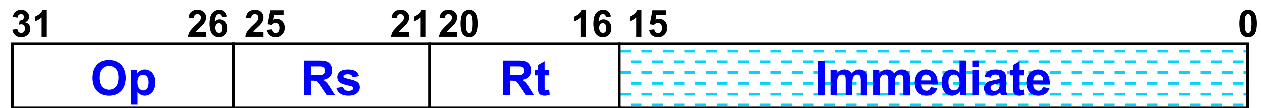
Operand Addressing: Register direct

Example: ADD $1, $2, $3 # $1 = $2 + $3



©GK Fall 2009

I-Type <op> rt, rs, immediate

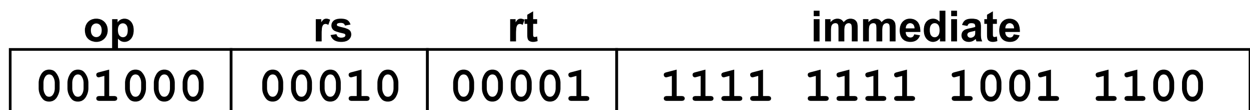


Immediate: 16 bit value

Operand Addressing: Register Direct and Immediate

Add Immediate Example

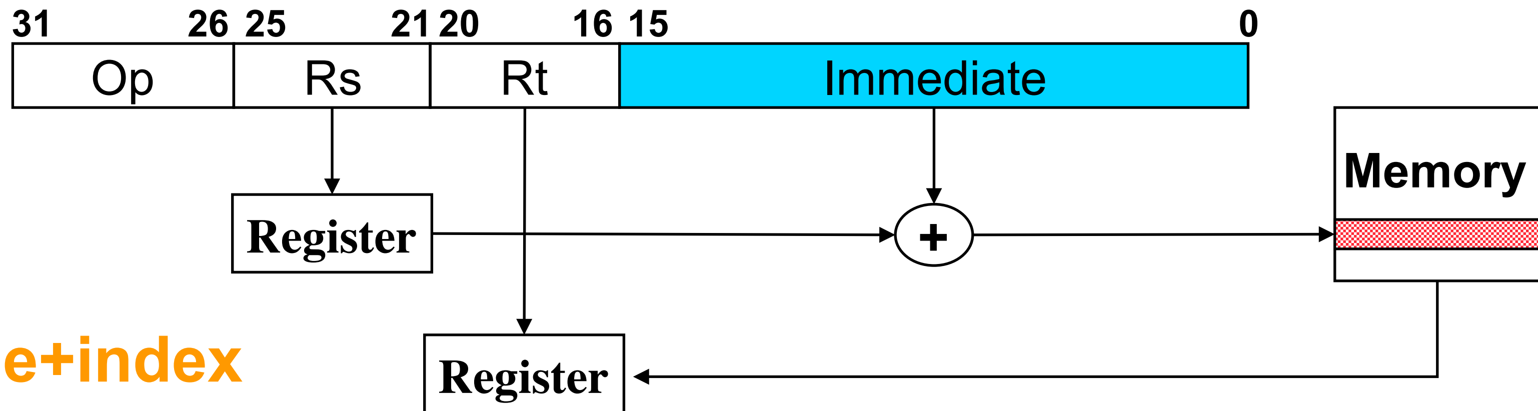
addi $1, $2, -100 # $1 = $2 + (-100)



Rt becomes the destination register!

©GK Fall 2009

I-Type <op> rt, rs, immediate



Base+index

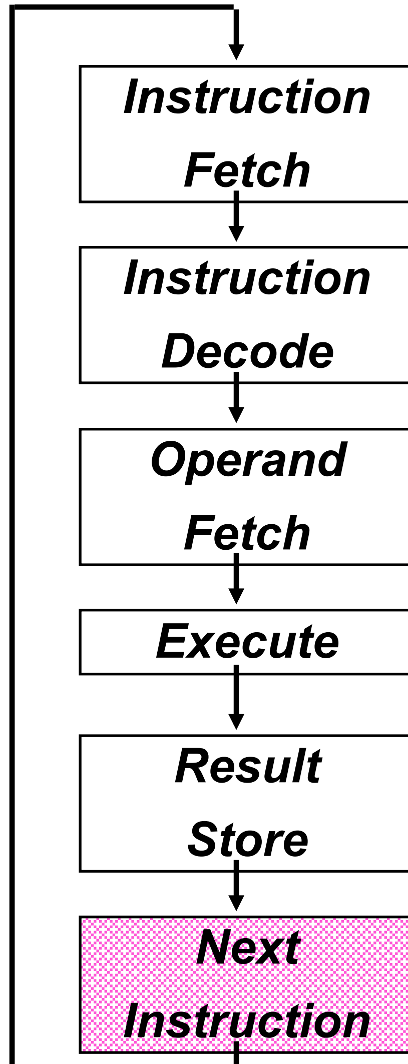
Load Word Example

lw $1, 100($2) # $1 = Mem[$2+100]

op	rs	rt	immediate
010011	00010	00001	0000 0000 0110 0100

©GK Fall 2009

Successor Instruction




```
main()
{
  int x,y,same;          // $0 == 0 always
  x = 43;                // addi $1, $0, 43
  y = 2;                 // addi $2, $0, 2
  same = 0;              // addi $3, $0, 0
  if (x == y)
        same = 1;        // execute only if x==y
                         // addi $3, $0, 1
}
```


©GK Fall 2009

The Program Counter

- Special register (pc) that points to instructions
- Contains memory address (like a pointer)
- Instruction fetch is
  - inst = mem[pc]
- To fetch next sequential instruction pc = pc + ?
  - Size of instruction?

©GK Fall 2009

The Program Counter

```
x = 43;          // addi $1, $0, 43

y = 2;           // addi $2, $0, 2

same = 0;        // addi $3, $0, 0

if (x == y) same = 1;  // addi $3, $0, 1 execute if x == y
```

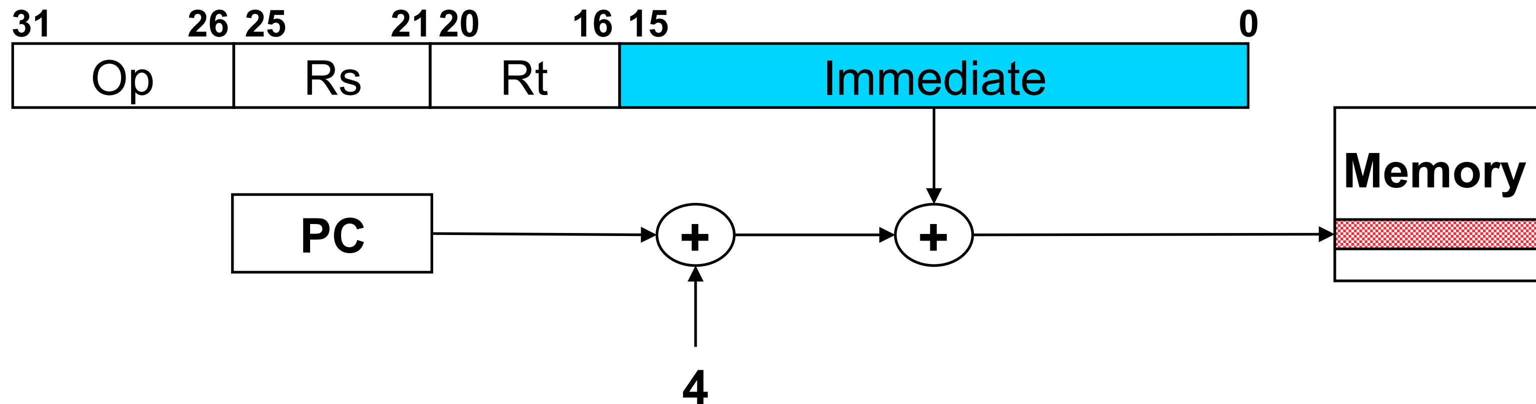
PC
0x10000
0x10004
0x10008
0x1000c

Address	Instruction
0x10000	addi $1, $0, 43
0x10004	addi $2, $0, 2
0x10008	addi $3, $0, 0
0x1000c	addi $3, $0, 1

Clearly, this is not correct
We cannot always execute both 0x10008 and 0x1000c

©GK Fall 2009

I-Type <op> rt, rs, immediate

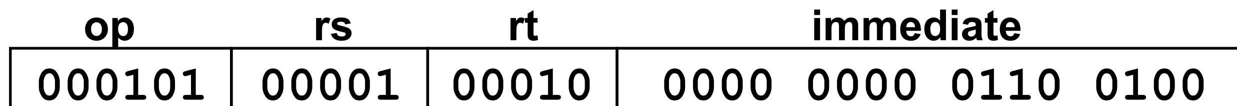


- PC relative addressing

Branch Not Equal Example

bne $1, $2, 100 # If ($1!= $2) goto [PC+4+100]

- +4 because by default we increment for sequential
  - more detailed discussion later in semester



op	rs	rt	immediate
000101	00001	00010	0000 0000 0110 0100

©GK Fall 2009

The Program Counter

```
x = 43;        // addi $1, $0, 43

y = 2;         // addi $2, $0, 2

if (x == y)

  same = 1;  // addi $3, $0, $1 execute if x == y
```

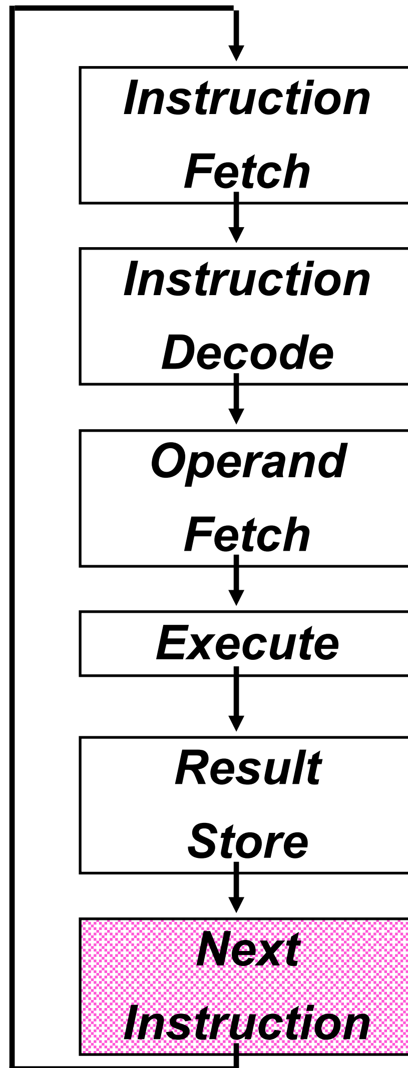
PC
0x10000
0x10004
0x10008
0x1000c
0x10010

Address	Instruction
0x10000	addi $1, $0, 43
0x10004	addi $2, $0, 2
0x10008	addi $3, $0, 0
0x1000c	bne $1, $2, 4
0x10010	addi $3, $0, 1

Understand branches

©GK Fall 2009

Successor Instruction




```
int equal(int a1, int a2) {
   int tsame;
   tsame = 0;
   if (a1 == a2)
           tsame = 1;    // only if a1 == a2
   return(tsame);
}
main()
{
   int x,y,same;      // r0 == 0 always
   x = 43;            // addi $1, $0, 43
   y = 2;             // addi $2, $0, 2
   same = equal(x,y); // need to call function
   // other computation
}
```


©GK Fall 2009

The Program Counter

- Branches are limited to 16 bit immediate
- Big programs?

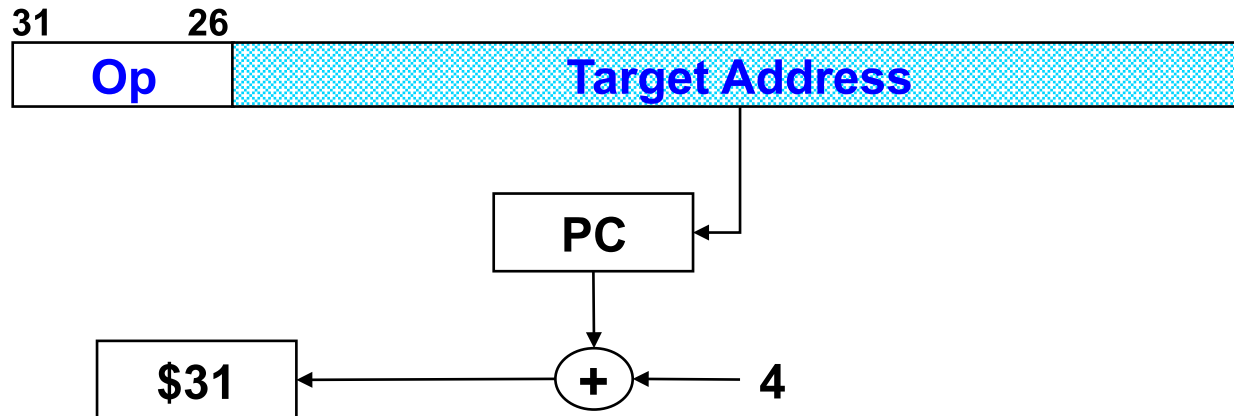
```
x = 43;  // addi $1, $0, 43

y = 2;   // addi $2, $0, 2

same = equal(x,y);
```

Address	Instruction
0x10000	`addi $1, $0, 43`
0x10004	`addi $2, $0, 2`
0x10008	“go execute equal”

Address	Instruction
0x30408	`addi $3, $0, 0`
0x3040c	`beq $1, $2, 8`
0x30410	`addi $3, $0, 1`
	“return $3”

©GK Fall 2009

J-Type: <op> target



Jump and Link Example

jal 0x0fab8 # PC<- 0x0fab8, $31<-PC+4

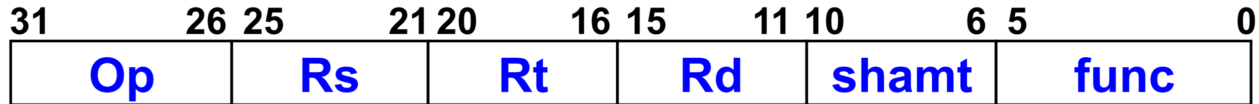
$31 set as side effect, used for returning, implicit operand

op	Target
000011	00 0000 0000 0011 1110 1010 1110

Please note, The address is a WORD ADDRESS!

©GK Fall 2009

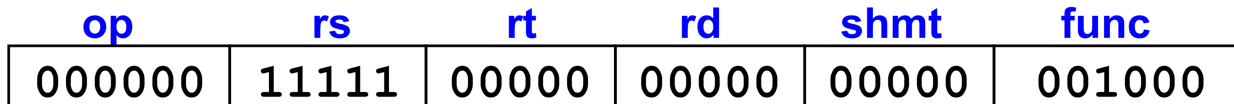
R Type: <OP> rd, rs, rt



31 – 26	25 – 21	20 – 16	15 – 11	10 – 6	5 – 0
Op	Rs	Rt	Rd	shamt	func

Jump Register Example

jr $31 # PC <- $31



op	rs	rt	rd	shmt	func
000000	11111	00000	00000	00000	001000

©GK Fall 2009

Instructions for Procedure Call and Return

```
int equal(int a1, int a2) {
   int tsame;
   tsame = 0;
   if (a1 == a2)
        tsame = 1;     return
   (tsame);
}
main()
{
   int x,y,same;
   x = 43;
   y = 2;
   same = equal(x,y);
   // other computation
}
```

Address	Instruction
0x10000	addi $1, $0, 43
0x10004	addi $2, $0, 2
0x10008	jal 0x30408
0x1000c	??

Address	Instruction
0x30408	addi $3, $0, 0
0x3040c	bne $1, $2, 4
0x30410	addi $3, $0, 1
0x30414	jr $31

PC	$31
0x10000	??
0x10004	??
0x10008	??
0x30408	0x1000c
0x3040c	0x1000c
0x30410	0x1000c
0x30414	0x1000c
0x1000c	0x1000c

©GK Fall 2009

MIPS Arithmetic Instructions

Instruction	Example	Meaning	Comments
add	add $1,$2,$3	$1 = $2 + $3	3 operands
subtract	sub $1,$2,$3	$1 = $2 – $3	3 operands
add immediate	addi $1,$2,100	$1 = $2 + 100	+ constant
add unsigned	addu $1,$2,$3	$1 = $2 + $3	3 operands
subtract unsigned	subu $1,$2,$3	$1 = $2 – $3	3 operands
add imm. unsign.	addiu $1,$2,100	$1 = $2 + 100	+ constant
multiply	mult $2,$3	Hi, Lo = $2 x $3	64-bit signed product
multiply unsigned	multu $2, $3	Hi, Lo = $2 x $3	64-bit unsigned product
divide	div $2,$3	Lo = $2 ÷ $3,	Lo = quotient,
		Hi = $2 mod $3	Hi = remainder
divide unsigned	divu $2,$3	Lo = $2 ÷ $3,	Unsigned quotient
		Hi = $2 mod $3	Unsigned remainder
Move from Hi	mfhi $1	$1 = Hi	Used to get copy of Hi
Move from Lo	mflo $1	$1 = Lo	Used to get copy of Lo

Which add for address arithmetic? Which for integers?

©GK Fall 2009

MIPS Logical Instructions

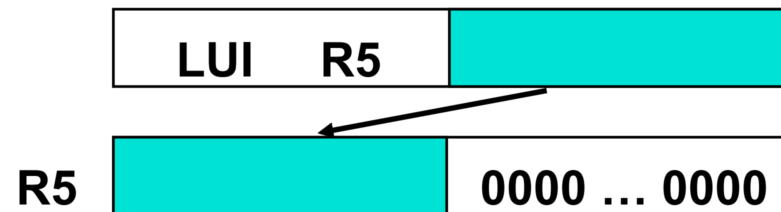
<table>
<tr><th>Instruction</th><th>Example</th><th>Meaning</th><th>Comment</th></tr>
<tr><td>and</td><td>and $1,$2,$3</td><td>$1 = $2 & $3</td><td>Bitwise AND</td></tr>
<tr><td>or</td><td>or $1,$2,$3</td><td>$1 = $2 | $3</td><td>Bitwise OR</td></tr>
<tr><td>xor</td><td>xor $1,$2,$3</td><td>$1 = $2 ⊕ $3</td><td>Bitwise XOR</td></tr>
<tr><td>nor</td><td>nor $1,$2,$3</td><td>$1 = ~($2 | $3)</td><td>Bitwise NOR</td></tr>
<tr><td>and immediate</td><td>andi $1,$2,10</td><td>$1 = $2 & 10</td><td>Bitwise AND reg, const</td></tr>
<tr><td>or immediate</td><td>ori $1,$2,10</td><td>$1 = $2 | 10</td><td>Bitwise OR reg, const</td></tr>
<tr><td>xor immediate</td><td>xori $1, $2,10</td><td>$1 = ~$2 &~10</td><td>Bitwise XOR reg, const</td></tr>
<tr><td>shift left logical</td><td>sll $1,$2,10</td><td>$1 = $2 << 10</td><td>Shift left by constant</td></tr>
<tr><td>shift right logical</td><td>srl $1,$2,10</td><td>$1 = $2 >> 10</td><td>Shift right by constant</td></tr>
<tr><td>shift right arithm.</td><td>sra $1,$2,10</td><td>$1 = $2 >> 10</td><td>Shift right (sign extend)</td></tr>
<tr><td>shift left logical</td><td>sllv $1,$2,$3</td><td>$1 = $2 << $3</td><td>Shift left by var</td></tr>
<tr><td>shift right logical</td><td>srlv $1,$2, $3</td><td>$1 = $2 >> $3</td><td>Shift right by var</td></tr>
<tr><td>shift right arithm.</td><td>srav $1,$2, $3</td><td>$1 = $2 >> $3</td><td>Shift right arith. by var</td></tr>
</table>

©GK Fall 2009

MIPS Data Transfer Instructions

Instruction	Comment
SW R3, 500(R4)	Store word
SH R3, 502(R2)	Store half
SB R2, 41(R3)	Store byte
LW R1, 30(R2)	Load word
LH R1, 40(R3)	Load half word
LHU R1, 40(R3)	Load half word unsigned
LB R1, 40(R3)	Load byte
LBU R1, 40(R3)	Load byte unsigned
LUI R1, 40	Load Upper Immediate (16 bits shifted left by 16)

Why do we need LUI?



©GK Fall 2009

MIPS Compare and Branch

Compare and Branch

Instruction	Operands	Meaning
beq	rs, rt, offset	if R[rs] == R[rt] then PC-relative branch
bne	rs, rt, offset	<>

Compare to zero and Branch

Instruction	Operands	Meaning
blez	rs, offset	if R[rs] <= 0 then PC-relative branch
bgtz	rs, offset	>
bltz	rs, offset	<
bgez	rs, offset	>=
bltzal	rs, offset	if R[rs] < 0 then branch and link (into R 31)
bgeal	rs, offset	>=

- Remaining set of compare and branch take two instructions
- Almost all comparisons are against zero!

©GK Fall 2009

MIPS jump, branch, compare instructions

Instruction	Example	Meaning
branch on equal	beq $1,$2,100	if ($1 == $2) go to PC+4+100 Equal test; PC relative branch
branch on not eq.	bne $1,$2,100	if ($1!= $2) go to PC+4+100 Not equal test; PC relative
set on less than	slt $1,$2,$3	if ($2 < $3) $1=1; else $1=0 Compare less than; 2's comp.
set less than imm.	slti $1,$2,100	if ($2 < 100) $1=1; else $1=0 Compare < constant; 2's comp.
set less than uns.	sltu $1,$2,$3	if ($2 < $3) $1=1; else $1=0 Compare less than; natural numbers
set l. t. imm. uns.	sltiu $1,$2,100	if ($2 < 100) $1=1; else $1=0 Compare < constant; natural numbers
jump	j 10000	go to 10000 Jump to target address
jump register	jr $31	go to $31 For switch, procedure return
jump and link	jal 10000	$31 = PC + 4; go to 10000 For procedure call

©GK Fall 2009

Signed v.s Unsigned Comparison

R1= 0…00 0000 0000 0000 0001

R2= 0…00 0000 0000 0000 0010

R3= 1…11 1111 1111 1111 1111

- **After executing these instructions:**

```
slt  r4,r2,r1
slt  r5,r3,r1
sltu r6,r2,r1
sltu r7,r3,r1
```

- **What are values of registers r4 - r7? Why?**

r4 = ; r5 = ; r6 = ; r7 = ;

©GK Fall 2009

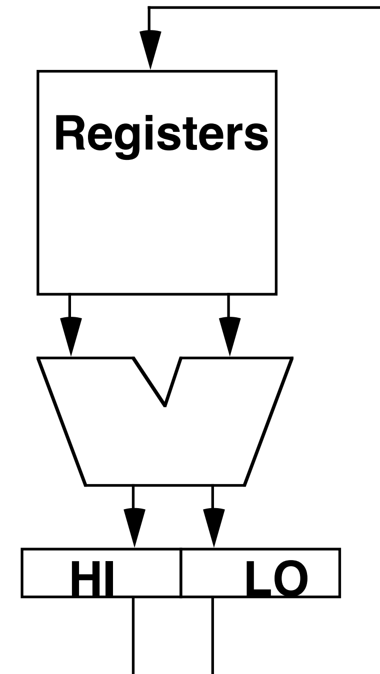
Multiply / Divide

Start multiply, divide

mult rs, rt

mfhi rd — Move from HI or LO

mflo rd



Why not Third field for destination?
(Hint: how many clock cycles for multiply or divide vs. add?)

©GK Fall 2009

Summary

- MIPS has 5 categories of instructions
  - Arithmetic, Logical, Data Transfer, Conditional Branch, Unconditional Jump
- 3 Instruction Formats: R-type, I-type, J-type.

Next:

- Assembly Programming

Reading

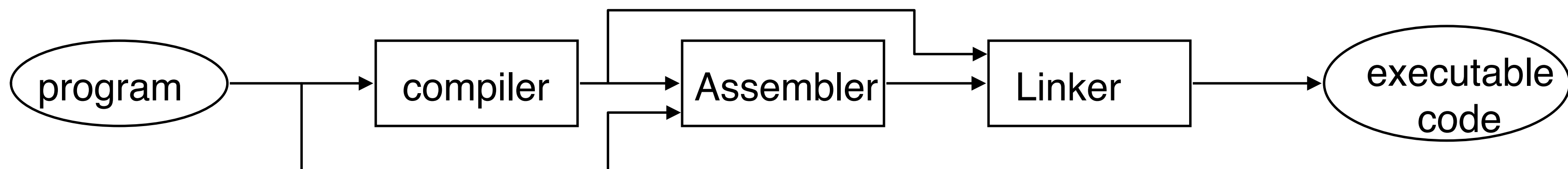
- Ch. 2, Appendix A

©GK Fall 2009

Assembler and Assembly Language

- Machine language is a sequence of binary words.

- **Assembly language** is a **text representation** for machine language plus extras that make assembly language programming easier (more readable too!).



©GK Fall 2009

The SPIM Simulator

- SPIM is a simulator that let you run and debug MIPS assembler programs.
- XSPIM is an X-window version of the simulator.
- The simulator allows you to look at the content of registers and memory, and to single step through the simulation.
- Documentation in postscript format and PDF format are in:
  http://kedem.cs.duke.edu/cps104/Handouts.html
- Also available a MAN page for SPIM.
- Try to run the provided example of an simple assembler program:
  http://kedem.cs.duke.edu/cps104/Handouts/MIPS.asm

©GK Fall 2009

Assembly Language

- One instruction per line.
- Numbers: are base-10 integers or Hex.
- Identifiers: alphanumeric, _, string starting in a letter or _
- Labels: identifiers starting at the beginning of a line followed by ":"
- Comments: everything following # till end-of-line.
- Instruction format: Space and "," separated fields.
  - [Label:] <op> Arg1, [Arg2], [Arg3] [# comment]
  - [Label:] <op> arg1, offset(reg) [# comment]
  - .Directive [arg1], [arg2], . . .

©GK Fall 2009

Assembly Language (cont.)

- Pseudo-instructions: extending the instruction set for convenience.

- Examples:
  - move $2, $4 — # $2 = $4, (copy $4 to $2)
    Translates to:
    add $2, $4, $0
  - li $8, 40 — # $8 = 40, (load 40 into $8)
    addi $8, $0, 40
  - sd $4, 0($29) — # mem[$29] = $4; Mem[$29+4] = $5
    sw $4, 0 ($29)
    sw $5, 4($29)
  - la $4, 0x1000056c — # Load address $4 = <address>
    lui $4, 0x1000
    ori $4, $4, 0x056c

©GK Fall 2009

Assembly Language (cont.)

- **The assembler is "smart enough" to figure out what instructions to use depending on the arguments' types.**
  - **For Example:**
  - `add $1, $2, -345` **will be translated into** `addi $1, $2, -345`
  - `div  $1, $2, $3` **will be translated into**

```
div  $2, $3
mflo $1
```

- **The assembler also translates labels into addresses.**
- **For Example one can use:**
  - `la $1, array1` **to load the address that the label** `array1` **is attached to.**
  - `bne  $2, $4, loop1` **to branch into the instruction that has the label** `loop1`

©GK Fall 2009

Assembly Language; `Directives`

- **Directives:** `“.”<string> [arg1], [arg2] . . .`
- **Examples:**
  - `.align n` # align datum on 2^n byte boundary.
  - `.ascii “<string>”` # store a string in memory.
  - `.asciiz “<string>”` # store a null terminated string in memory
  - `.data [address]` # start a data segment.
    # [optional beginning address]
  - `.text [address]` # start a code segment.
  - `.word w1, w2, . . . , wn` # store n words in memory.

©GK Fall 2009

The routine written in C

```
#include <stdio>

int main ( )
{
        int i;
        int sum = 0;
        for(i=0; i <= 100; i++) sum = sum + i*i ;
        printf("The sum from 0 .. 100 is %d\n", sum) ;
}
```


©GK Fall 2009

Assembly Language Example1:

```
        .text
        .align  2
main:
        la      $10, Temp
loop:
        lw      $14, 4($10)
        mul     $15, $14, $14
        lw      $24, 0($10)
        add     $25, $24, $15
        sw      $25, 0($10)
        addi    $8, $14, 1
        sw      $8, 4($10)
        ble     $8, 100, loop
        la      $4, str
        lw      $5, 0($10)
        jal     printf
```

```
        li      $2, 10
        syscall                 # Exit

        .data
        .align 2
Temp:   .word 0, 0
str:
   .asciiz "The sum from 0 .. 100 is
%d\n"
```


©GK Fall 2009